Bflow* Toolbox

Dokumentation

Anpassen und Erweitern von Bflow*

www.bflow.org

Inhalt

Inhalt 2
Zielgruppe dieses Handbuchs 3
Erweiterung der Validierungsregeln für die Live Validation 4
Attribute von Regeln 4
Prüflogik mittels Check 6
Prüflogik mittels Epsilon 6
Erweiterung der Validierungsregeln für die erweiterte Modellprüfung 8
Hinzufügen eines eigenen Exportformats 9
Hinzufügen eines eigenen Importformats 11
Die Add-on-Schnittstelle 13
Welche Programme lassen sich einbinden? 15
MTG-Sim – Eine Beispielprotokoll 15
Ein eigenes Prolog-Programm einrichten 21
Programmparameter 25
Prologparameter 26
Startbefehlskonvention 27
Add-on-Komponenten 27
Formkonvention für Analysen und Modifikationen des Modells 30
Arbeit mit externen Dateien 32

Zielgruppe dieses Handbuchs

Dieses Handbuch richtet sich an Administratoren mit Programmiererfahrung, die den Leistungsumfang von Bflow* erweitern wollen. An einigen Stellen bietet die Bflow* Toolbox hierfür einfache Schnittstellen.

Es ist jedoch nicht Ziel dieses Handbuchs, eine vollständige Dokumentation für Entwickler zu liefern.

Endanwender werden mit den hier vorgestellten Erweiterungen nicht direkt in Berührung kommen; für sie genügt die Lektüre des Bflow*-Benutzerhandbuchs.

Erweiterung der Validierungsregeln für die Live Validation

Mit "Live Validation" bietet Bflow* die Möglichkeit, die Korrektheit von erstellten Modellen während der Eingabe zu überprüfen.

Die hier betrachteten Regeln betreffen die Logik im Metamodell und die dargestellten Objekte im EPC- bzw. OEPC-Diagramm.

Eine Regel besteht im Allgemeinen aus zwei Teilen: Der eigentlichen Prüflogik und Beschreibung der Regel.

Die Prüflogik wird im Moment von zwei unterschiedlichen Sprachen unterstützt: „Check“ und „Epsilon“. Beide sind als Eclipse-Plugin verfügbar. „Check“ ist ein Teil des Xtend- bzw. Xtext-Plugins[1], welches mit dem EMF-Plugin ab Eclipse-Version 3.5 ausgeliefert wird. „Epsilon“ ist ein eigenständiges Plugin von GMT[2].

Die Beschreibung der Regel erfolgt im xml-Format. Es gibt verschiedene Attribute um eine Regel zu beschreiben. Näheres hierzu später.

Generelle Festlegung:

Existiert für eine Regel eine Prüflogik, aber keine Beschreibung, so wird die Regel stets angewendet und man kann sie nicht abschalten.

Attribute von Regeln

Eine Regel besitzt die folgenden Attribute:

Id	*Wird zur Identifizierung der Regel benutzt und muss innerhalb eines Diagrammtyps (EPC oder OEPC) eindeutig sein.*
Name	*Name der Regel der im Preferences-Dialog erscheint. Dieser kann beliebig gewählt werden, sollte aber nachvollziehbar sein.*
Message	*Dieses Attribut legt fest, wie die Fehlermeldung aussieht, wenn sie erscheinen sollte. Fügt man in den Meldungstext den Wert „$value“ ein, so wird dieser durch einen entsprechenden Parameter ersetzt.*
Class	*Gibt die Art bzw. die Klasse der Regel an. Dies kann beliebig gewählt werden und dient nur zur besseren Zuordnung von Regeln im Preferences-Dialog.*

1 http://www.eclipse.org/workinggroups/oaw/
2 GMT - http://www.eclipse.org/gmt/epsilon/

Description	*Dies ist ein Beschreibungstext, der die Regel kurz näher beschreiben sollte.*
Image	*An dieser Stelle kann ein relativer Pfad zu einem Bild eingefügt werden, dass die Regel symbolisiert.*
Default	*Gibt an, ob die Regel standardmäßig an- oder ausgeschaltet ist*

Ein simples Beispiel für eine derartige Regelbeschreibung soll hier gegeben werden:

```
<?xml version="1.0" encoding="UTF-8"?>
<language value="de">
      <rule>
            <id>SyntaxEventRule1</id>
            <name>EventRule1</name>
            <message>Ereignis $value hat mehr als einen eingehenden
                     Kontrollfluss. Eingehende Kontrollfluesse ueber Konnektoren
                     synchronisieren.
            </message>
            <class>Syntaxprüfung</class>
            <description>Prüft, ob das Ereignis mehr als einen eingehenden
                         Kontrollfluss hat.
            </description>
            <image></image>
            <default>true</default>
      </rule>
      <rule>
            <id>SyntaxEventRule2</id>
            <name>EventRule2</name>
            <message>Ereignis $value hat mehr als einen ausgehenden
                     Kontrollfluss. Ausgehende Kontrollfluesse ueber Konnektoren
                     synchronisieren.
            </message>
            <class>Syntaxprüfung</class>
            <description>Prüft, ob das Ereignis mehr als einen ausgehenden
                        Kontrollfluss hat.
            </description>
            <image></image>
            <default>true</default>
      </rule>
</language>
```

Die entsprechende Regelbeschreibung befindet sich im „org.bflow.toolbox.chk.syntax“-Plugin für EPC-Diagramme im Paket „language.epc“ und für OEPC-Diagramme im Paket „lanuage.oepc“.

Möchte man eine neue Regelbeschreibung hinzufügen, dann muss die entsprechende Datei dort nur ergänzt werden.

Prüflogik mittels Check

Wie bereits oben erläutert, ist Check ein Bestandteil des Xtend-/Xtext-Plugins. Lesen Sie bitte zu generellen Sprachspezifikation dort im Handbuch nach.

Ein Beispiel für eine implementierte EPC-Prüfregel kann so aussehen:

```
context epc::Event if(shallCheck("SyntaxEventRule1"))
      ERROR getErrorMessage("SyntaxEventRule1", this.name) :
      !(this.outgoingControlFlows().isEmpty &&
this.incomingControlFlows().isEmpty);
```

Zunächst erfolgt der Aufruf der Methode „shallCheck()“ um zu prüfen, ob diese Regel überhaupt angewendet werden soll. Das Anwenden der Regeln lässt sich im Preferences-Dialog festlegen. Als Parameter für den Aufruf wird die ID der Regel verwendet.

Soll die Regel angewendet werden und tritt in dem zu prüfenden Modell der entsprechend betrachtete Fehler auf, so wird die Methode „getErrorMessage()“ aufgerufen. Diese liefert die zu der Regel gehörende Fehlermeldung für den Anwender.

Der erste Parameter ist wieder die ID der Regel. Der zweite ist der Wert, der den „$value“ Teil im „message“ Attribut ersetzen soll.

Nach dem „:“ kommt dann die zu prüfende Logik.

Die Check-Regeln für EPK-Diagramme befinden im „org.bflow.toolbox.chk.syntax“-Plugin im Paket „check“ in den Dateien „epcSyntax.chk“ und „epcDeadlocks.chk“. Die erste Datei beinhaltet Regeln zur Syntaxprüfung, die zweite für den Kontrollfluss. Die Regeln für OEPC-Diagramme analog in den Dateien „oepcSyntax.chk“ und „oepcDeadlocks.chk“.

Prüflogik mittels Epsilon

Das Werkzeug Epsilon[3] besteht aus mehreren Sprachen für verschiedene Anwendungen. Für unsere Zwecke wird im speziellen die Sprache „evl“[4] bzw. „eol“[5] verwendet.

Für generelle Sprachspezifikation lesen Sie bitte dort im Handbuch nach.

Ein Beispiel für eine implementierte Prüflogik wird nun betrachtet:

[3] http://www.eclipse.org/gmt/epsilon/
[4] evl – Epsilon Validation Language
[5] eol – Epsilon Object Language

```
context BflowSymbol {

	critique HasName {

		guard : (self.isTypeOf(Event) or self.isTypeOf(BusinessObject))
		check : self.hasName() or not
				prefRequester.shallCheck('EpsilonSymbolRule1')
		message : prefRequester.getErrorMessage('EpsilonSymbolRule1',
							self.eClass().name)
		fix {
			title : 'Standardnamen vergeben'
			do {
				self.name := 'Element';
			}
		}

		fix
		{
			title : 'Prüfregel ausschalten'
			do { prefRequester.setRuleEnabled('EpsilonSymbolRule1',
								false); }
		}
	}
}
```

Diese Regel prüft alle Elemente vom Typ „BflowSymbol“ darauf, ob sie einen Namen haben.

Der Bereich „check“ beinhaltet dabei die Prüflogik und die Abfrage, ob diese Regel angewendet werden soll. Dies übernimmt die Methode „prefRequester.shallCheck()“. Das Anwenden der Regeln läßt sich im Preferences-Dialog festlegen.

Als Parameter für den Aufruf wird die ID der Regel verwendet.

Soll die Regel angewendet werden und tritt der entsprechend betrachtete Fehler auf, so wird die Methode „prefRequester.getErrorMessage()“ aufgerufen. Diese liefert die zu der Regel gehörende Fehlermeldung für den Anwender.

Der erste Parameter ist wieder die ID der Regel. Der zweite ist der Wert, der den „$value“ Teil im „message“ Attribut ersetzen soll.

Ein weiterer interessanter Punkt sind die „fix“-Bereiche. Diese dienen dazu, dem Anwender sog. „Quick-Fix“-Optionen anzubieten, welche aufgetretenen Fehler standardmäßig löst.

Die zweite Fix-Option schaltet die Prüfregel generell ab. Dies übernimmt der Methodenaufruf „prefRequester.setRuleEnabled()“.
Der erste Parameter ist die ID der Regel und der zweite ob die Regel ein- oder ausgeschaltet werden soll. Im Normalfall sollte hier „false“ stehen, da dieser Fix die Regel abschalten soll.

Die Regeln insgesamt befinden sich „org.bflow.toolbox.oepc.evl“-Plugin.

Das Paket „epc“ beinhaltet die Regeln für die EPC-Diagramme („epcSyntax.evl“) und der Ordner „validation“ die Regeln für die OEPC-Diagramme („syntax.evl“).

Erweiterung der Validierungsregeln für die erweiterte Modellprüfung

Um eigene Regeln für die erweiterte Modellprüfung hinzuzufügen, kann das Prolog-Programm im Plugin org.bflow.toolbox.mitamm erweitert werden.

Dieses befindet sich im Pfad org/bflow/toolbox/mitamm/prolog/pls

Beim Start der Modellprüfung wird das Prolog-Prädikat

a.

aufgerufen, das seinerseits alle Prüfungen startet.

Bei einem erkannten Problem muss das Prädikat

message(String1,String2,String3)

eingebunden werden.

Hierbei ist:

String1 – eine der Zeichenketten 'ERROR', 'WARNING' oder 'INFO', womit bestimmt wird, in welcher Form die Meldung in der Problems-View von Eclipse angezeigt wird

String2 – die ID desjenigen Modellelements, das beim Auftreten der Meldung markiert werden soll

String3 – der auszugebende Meldungstext als String

Hinzufügen eines eigenen Exportformats

Der Export in andere Diagrammarten wird mit Hilfe von sog. „XSLT-Skripts“ realisiert. Möchten Sie Bflow* um Ihr eigenes Importformat erweitern, so müssen Sie diese zunächst programmieren. Die Programmierung der XSLT-Skripts selbst wird in diesem Handbuch jedoch nicht behandelt.

Da ein Export auch aus der Hintereinanderausführung von solchen Skripts bestehen kann oder manche Skript Parameter benötigen, ist eine Datei erforderlich, die den gesamten Vorgang beschreibt. Diese Datei nennt sich „Export-Beschreibung“ und endet auf „exd.xml“.

Als Beispiel soll nun ein „Fiktiver Export“ gestaltet werden.

(*1.) Schritt: Anlegen einer Importbeschreibungsdatei*

Öffnen Sie mit einem XML-Editor oder einem normalen Texteditor eine neue Datei und nennen Sie sie „fiktiver export.exd.xml“. Bitte beachten Sie die korrekte Kennzeichnung der Endung! Erstellen Sie anschließend die Datei nach folgendem Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<exportdescription>
      <config>
            <name>Fiktiver Export</name>
            <fileextension>xml;fi</fileextension>
            <description>Exportiert ein Bflow*-Diagramm als fiktives Diagramm
            </description>
            <callstack>
                  <script>
                        <file>/scripts/fiExport1.xslt</file>
                  </script>
                  <script>
                        <file>/scripts/fiExport2.xslt</file>
                  </script>
                  <script>
                        <file>/scripts/fiResize.xslt</file>
                        <param Multiplikator="2" Streckung="3"/>
                  </script>
            </callstack>
      </config>
</exportdescription>
```

Die Attribute „name“, „fileextension“ und „description“ sind Pflichtangaben. Sie geben eine äußere Beschreibung des Vorgangs an, die u.a. im Exportdialog verwendet werden. Wenn eine der Angaben fehlt, gibt es zwar keine Fehlermeldung, dennoch wird der Export nicht ordnungsgemäß durchgeführt.

Möchten Sie als Quelldatei auch mehrere Typen zulassen, so geben Sie die Dateiendungen durch „;“ getrennt hintereinander an. Im dargestellten Beispiel behandelt der Export nur Dateien, die auf „.xml“ oder „.fi“ enden.

Im Abschnitt „callstack“ wird dann aufgelistet, welche Skripts der Reihe nach aufgerufen werden. Die Reihenfolge der aufgelistete Skripts in der Konfigurationsdatei, gibt auch die Aufrufreihenfolge beim Export vor.

Das Attribut „file“ gibt dabei den Ort des XSLT-Skripts relativ zur Konfigurationsdatei an.

Sollen für einen Skriptaufruf Parameter verwendet werden, so muss man das Attribut „param“ nutzen. Die Verwendung von Parametern ist, wie oben zu sehen, optional.

(2.) Schritt: Kopieren der benötigten Dateien

Nachdem Sie die Exportbeschreibung angelegt haben, müssen Sie sie in Ihrem Workspace in dem Unterverzeichnis „.export“ speichern. Bflow* registriert nur manuell hinzugefügte Exporte, wenn sie in diesem Verzeichnis liegen. Sollte es noch nicht existieren, dann legen Sie es an. Das könnte dann zum Beispiel so aussehen: „D:\bflow\workspace\.export\fiktiver export.exd.xml“.

Kopieren Sie anschließend die benötigten XSLT-Skripts zu der Stelle, die Sie in der Exportbeschreibung angegeben haben. Nach unserem Beispiel müssen die drei Skripts dann im Verzeichnis „D:\bflow\workspace\.export\scripts\“ liegen.

Wenn Sie nun den Bflow*-Export in Ihrer Arbeitsumgebung aufrufen, so sehen Sie den neuen Export aufgelistet:

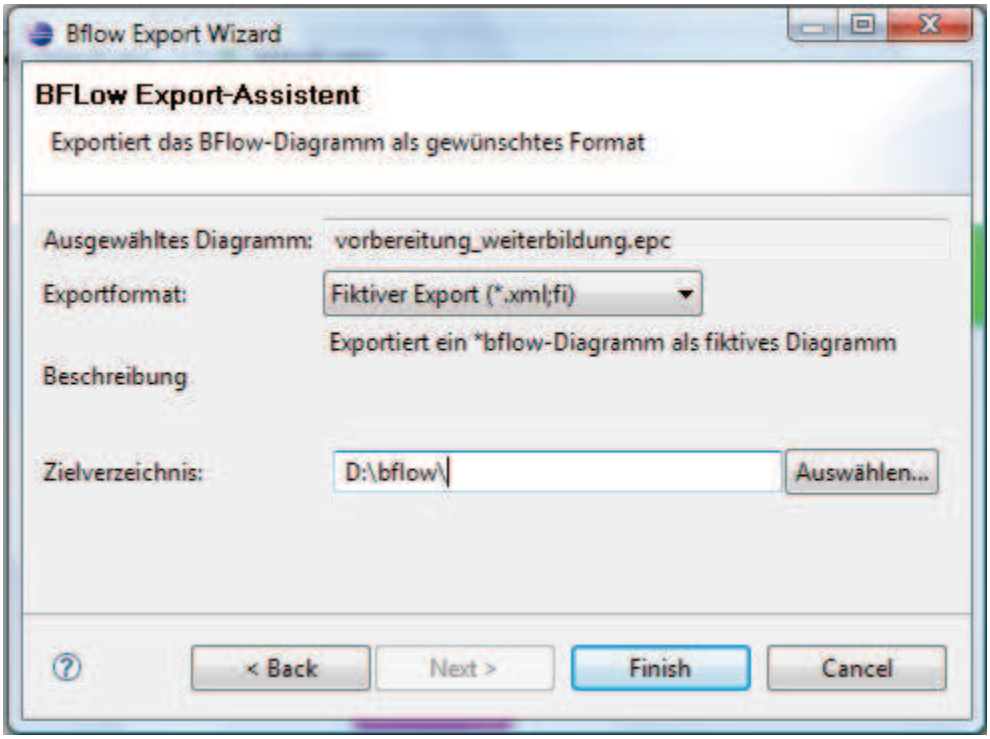


Abbildung 1: Der neue bflow-Export*

Hinzufügen eines eigenen Importformats

Der Import anderer Diagrammarten wird mit Hilfe von sog. „XSLT-Skripts“ realisiert. Möchten Sie Bflow* um Ihr eigenes Importformat erweitern, so müssen Sie diese zunächst programmieren. Die Programmierung der XSLT-Skripts selbst wird in diesem Handbuch jedoch nicht behandelt.

Da ein Import auch aus der Hintereinanderausführung von solchen Skripts bestehen kann oder manche Skript Parameter benötigen, ist eine Datei erforderlich, die den gesamten Vorgang beschreibt. Diese Datei nennt sich „Import-Beschreibung“ und endet auf „ixd.xml“.

Als Beispiel soll nun ein „Fiktiver Import“ gestaltet werden.

(*1.) Schritt: Anlegen einer Importbeschreibungsdatei*

Öffnen Sie mit einem XML-Editor oder einem normalen Texteditor eine neue Datei und nennen Sie sie „fiktiver import.ixd.xml“. Bitte beachten Sie die korrekte Kennzeichnung der Endung!
Erstellen Sie anschließend die Datei nach folgendem Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<importdescription>
      <config>
            <name>Fiktiver Import</name>
            <fileextension>xml;fi</fileextension>
            <description>Importiert ein fiktives Diagramm als Bflow*-
Diagramm
            </description>
            <callstack>
                  <script>
                        <file>/scripts/fiImport1.xslt</file>
                  </script>
                  <script>
                        <file>/scripts/fiImport2.xslt</file>
                  </script>
                  <script>
                        <file>/scripts/fiResize.xslt</file>
                        <param Multiplikator="2" Streckung="3"/>
                  </script>
            </callstack>
      </config>
</importdescription>
```

Die Attribute „name“, „fileextension“ und „description“ sind Pflichtangaben. Sie geben eine äußere Beschreibung des Vorgangs an, die u.a. im Importdialog verwendet werden. Wenn eine der Angaben fehlt, gibt es zwar keine Fehlermeldung, dennoch wird der Import nicht ordnungsgemäß durchgeführt.

Möchten Sie als Quelldatei auch mehrere Typen zulassen, so geben Sie die Dateiendungen durch „;“ getrennt hintereinander an. Im dargestellten Beispiel behandelt der Import nur Dateien, die auf „.xml“ oder „.fi“ enden.

Im Abschnitt „callstack“ wird dann aufgelistet, welche Skripts der Reihe nach aufgerufen werden. Die Reihenfolge der aufgelistete Skripts in der Konfigurationsdatei, gibt auch die Aufrufreihenfolge beim Import vor.
Das Attribut „file“ gibt dabei den Ort des XSLT-Skripts relativ zur Konfigurationsdatei an.

Sollen für einen Skriptaufruf Parameter verwendet werden, so muss man das Attribut

„param“ nutzen. Die Verwendung von Parametern ist, wie oben zu sehen, optional.

(*2.) Schritt: Kopieren der benötigten Dateien*

Nachdem Sie die Importbeschreibung angelegt haben, müssen Sie sie in Ihrem Workspace in dem Unterverzeichnis „.import“ speichern. Bflow* registriert nur manuell hinzugefügte Importe, wenn sie in diesem Verzeichnis liegen. Sollte es noch nicht existieren, dann legen Sie es an. Das könnte dann zum Beispiel so aussehen: „D:\bflow\workspace\.import\fiktiver import.ixd.xml“.

Kopieren Sie anschließend die benötigten XSLT-Skripts zu der Stelle, die Sie in der Importbeschreibung angegeben haben. Nach unserem Beispiel müssen die drei Skripts dann im Verzeichnis „D:\bflow\workspace\.import\scripts\“ liegen.

Wenn Sie nun den Bflow*-Import in Ihrer Arbeitsumgebung aufrufen, so sehen Sie den neuen Import aufgelistet.

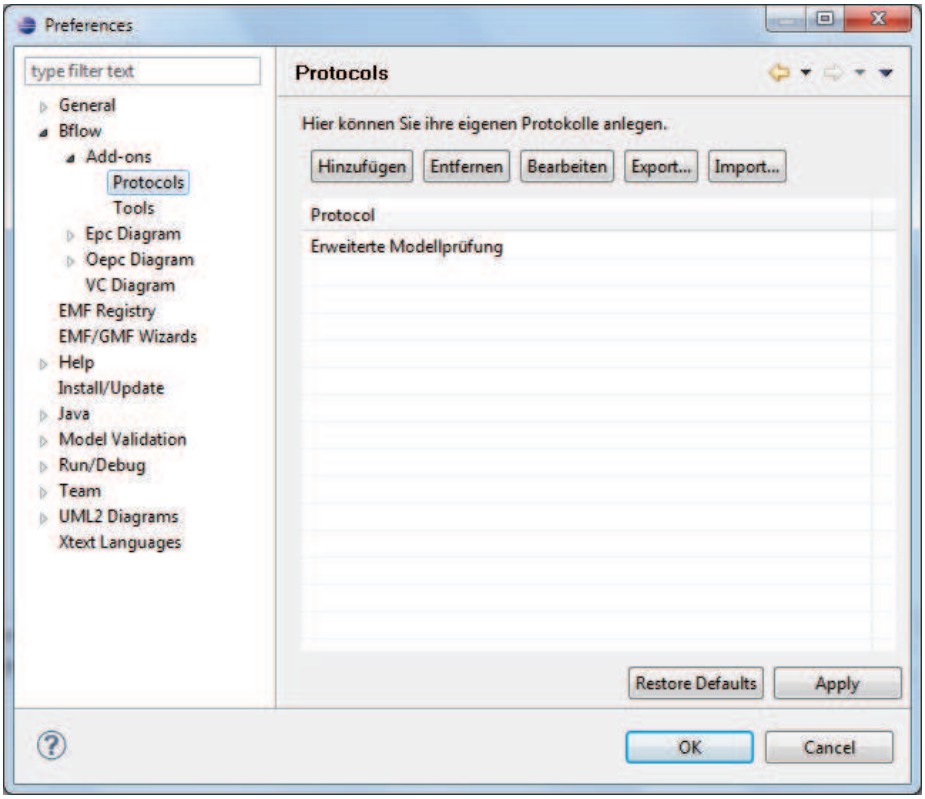
Die Add-on-Schnittstelle

Wie Sie schon von der Arbeit mit bflow* wissen, ist es möglich externe Programme in die Bearbeitung und Analyse von Modellen zu integrieren. Dieses Feature wird zum Beispiel von der erweiterten Modellprüfung und der EPC-Metrik-Analyse genutzt.

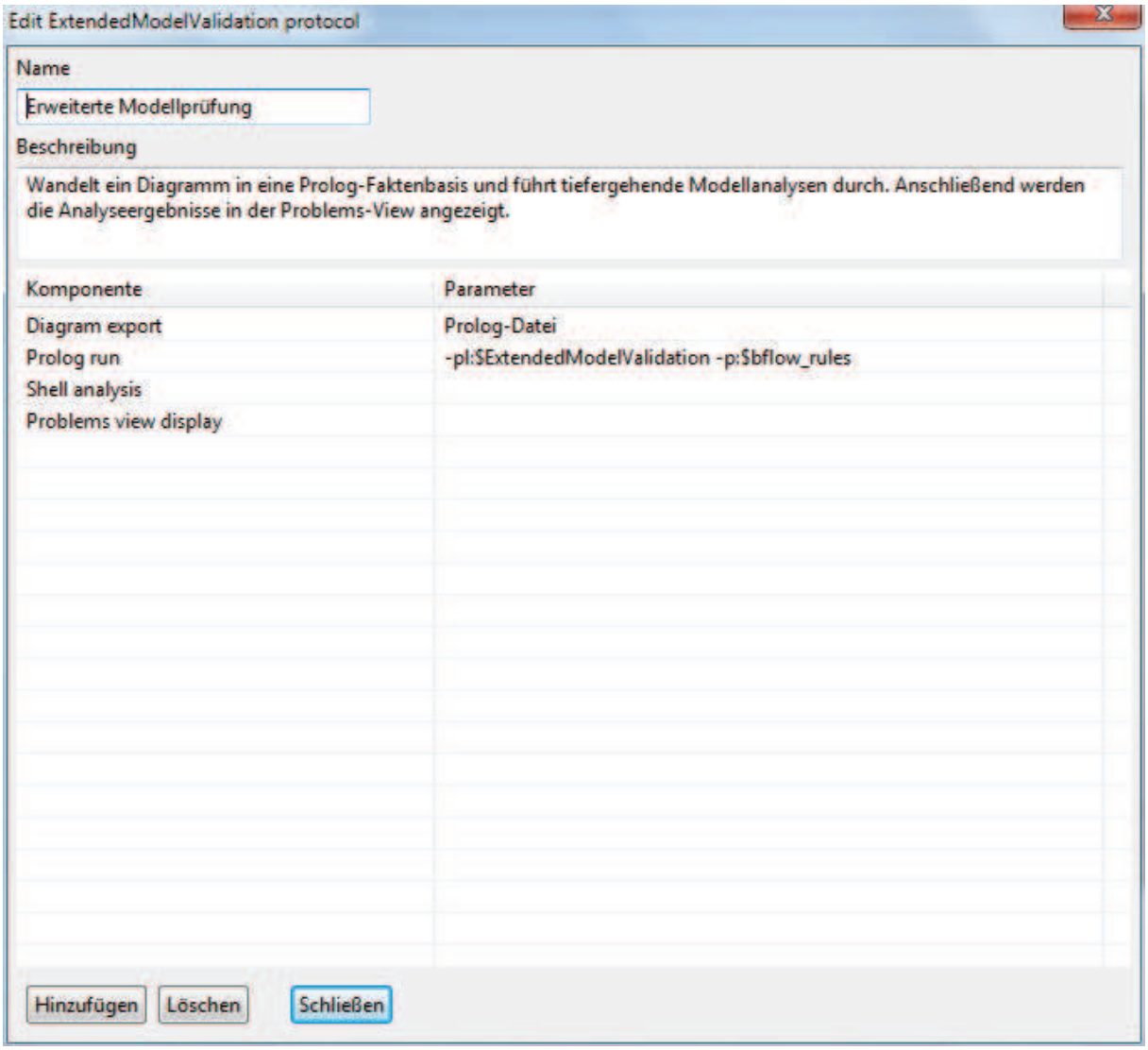
Neben den bereits bekannten Möglichkeiten gibt es noch weitere: So ist es zum Beispiel möglich, dass externe Programme Modell- und Elementattribute auslesen, zu bearbeiten und verändern. Zu diesen Attributen zählen nicht nur die, die sich per „Attribute-View“ hinzufügen lassen, sondern auch grafische Attribute wie Schriftart, -Form, -Stil oder Position.

Diese Features werden durch die Addon-Schnittstelle realisiert. Es handelt sich hierbei um eine modulare Schnittstelle für Programme außerhalb der bflow*-Programmumgebung.
Das Prinzip dabei ist denkbar einfach: Jede Interaktion mit einem externen Programm wird durch ein Protokoll geregelt. Dieses Protokoll bestimmt, welche Daten auf welche Weise kommuniziert werden.

Schauen wir uns dies zunächst am Beispiel der erweiterten Modellprüfung an: Öffnen Sie die folgenden Menüpunkte: „Window“ → „Preferences“. Selektieren Sie nun den Unterpunkt „Bflow“ → „Add-ons“ → „Protocols“. Sie sollten nun das folgende Fenster sehen:



Wählen Sie nun die „Erweiterte Modellprüfung“ und klicken Sie auf „Bearbeiten“. Es öffnet sich das folgende Fenster:



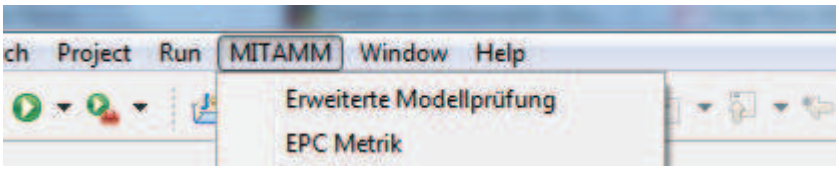
Dieses Fenster stellt das Konfigurationsmenü für ein Protokoll dar. Sie können hier den Namen, die Beschreibung und die Komponenten des Protokolls festlegen.

Wie Sie sehen, besteht das Protokoll aus 4 Komponenten. Eine Komponente stellt dabei einen Interaktionsschritt dar. Einige Komponenten besitzen Parameter. Andere wiederum nicht. Eine vollständige Auflistung aller Komponenten finden Sie am Ende des Kapitels.

Der Ablauf eines Protokolls richtet sich immer nach der Reihenfolge der Komponenten, von oben nach unten gelesen.
Anhand des oben dargestellten Beispiels lässt sich somit sehen, dass das Diagramm zuerst exportiert und dann anschließend Prolog gestartet wird. Nachdem Prolog seine Ausführung beendet hat, wird die Shell[6]-Ausgabe von Prolog analysiert. Die für bflow* relevanten Informationen werden dann extrahiert und auf der Problems-View dargestellt.

Gültige und ausführbare Protokolle werden Ihnen automatisch im Add-on-Menü von bflow* angezeigt.

[6] Externe Programme werden immer in einem eigenen Prozess gestartet und besitzen daher eine eigene Prozesslaufzeitumgebung. Diese Umgebung wird mit Shell bezeichnet.



Welche Programme lassen sich einbinden?

Über die Add-on-Schnittstelle lassen sich theoretisch alle Programme, die auf Ihrem Betriebssystem selbstständig lauffähig sind, ansprechen und einbinden. Dabei spielt es keine Rolle, ob es sich um eine exe-, jar- oder Shell-Datei handelt.

Lässt sich das Programm von Ihnen manuell entweder per Kommandozeile oder Doppelklick starten, so kann es auch von bflow* gestartet werden.

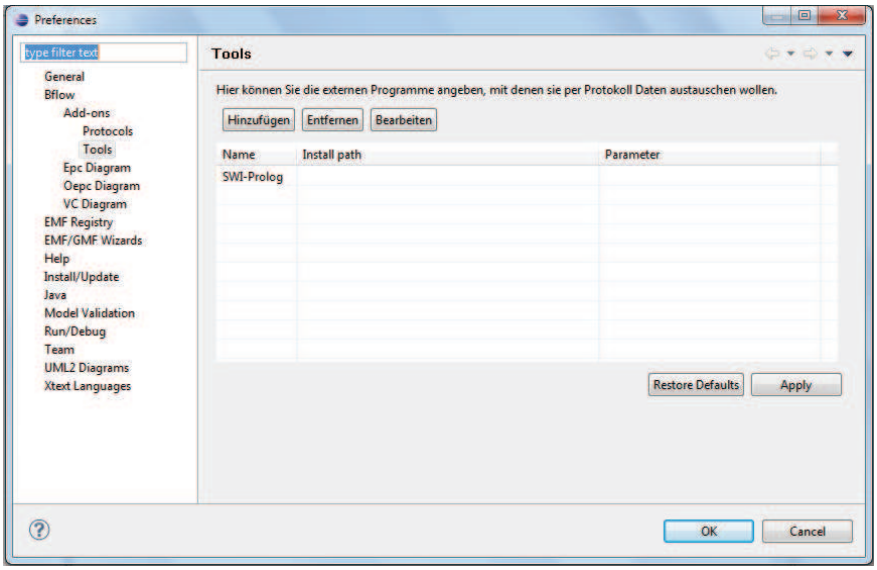
MTG-Sim – Eine Beispielprotokoll

Es soll nun erläutert werden, wie man selber ein eigenes Protokoll einrichtet. Das Programm, das dabei verwendet werden soll, heißt „MTG-Sim“. Das in Java geschriebene Programm liegt als „MTGSim.jar“ in einem beliebigen Verzeichnis auf der Festplatte.

Inhalt des Programms ist es, Modellabläufe zu simulieren. Anhand von gegebenen Wahrscheinlichkeiten und Ausführungszeiten simuliert das Programm einen Prozessablauf und misst dabei zum einen, wo die Simulation endete und zum anderen nach welcher Zeit. Die Ergebnisse werden abschließend von dem Programm in eine Datei geschrieben.

Die Aufgabe des einzurichtenden Protokolls ist es somit, das Modell in einem Format zu exportieren, das das Programm lesen kann, und nach der Analyse die Ergebnisse wieder an bflow* zurückzuführen.

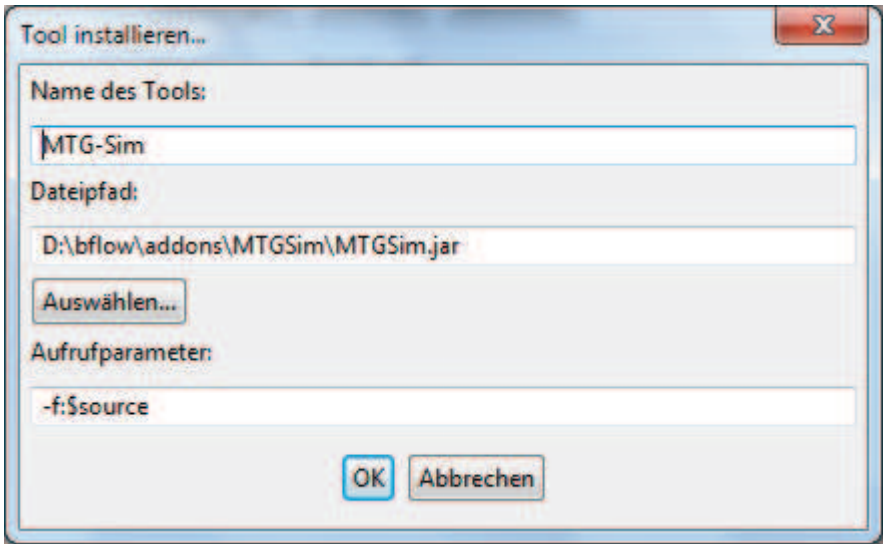
Zuerst muss das Programm als externes Werkzeug - auch Tool genannt – an der Add-on-Schnittstelle angemeldet werden. Dies geschieht wie folgt:
Öffnen Sie die Menüs „Window“ → „Preferences“. Wählen Sie nun die Punkte „Bflow“ → „Add-ons“ → „Tools. Sie sehen nun das folgende Fenster:



Klicken Sie auf den Button „Hinzufügen“ und tragen Sie in dem folgenden Dialog die entsprechenden Werte ein:

Name des Tools: „MTG-Simulator“
Dateipfad: *Setzen Sie hier den Pfad zur „MTGSim.jar“-Datei.*
Zum Beispiel: „D:\bflow\MTGSim.jar“
Aufrufparameter: „-f:$source“

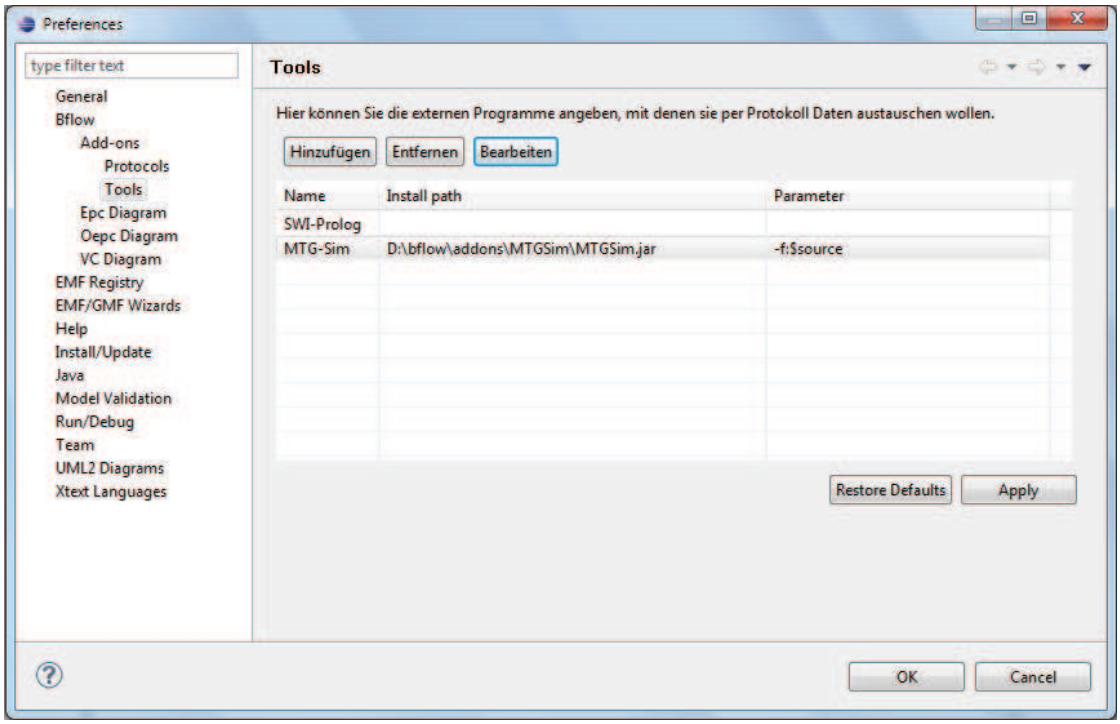
Das fertige Dialogfenster sollte in etwa so aussehen:



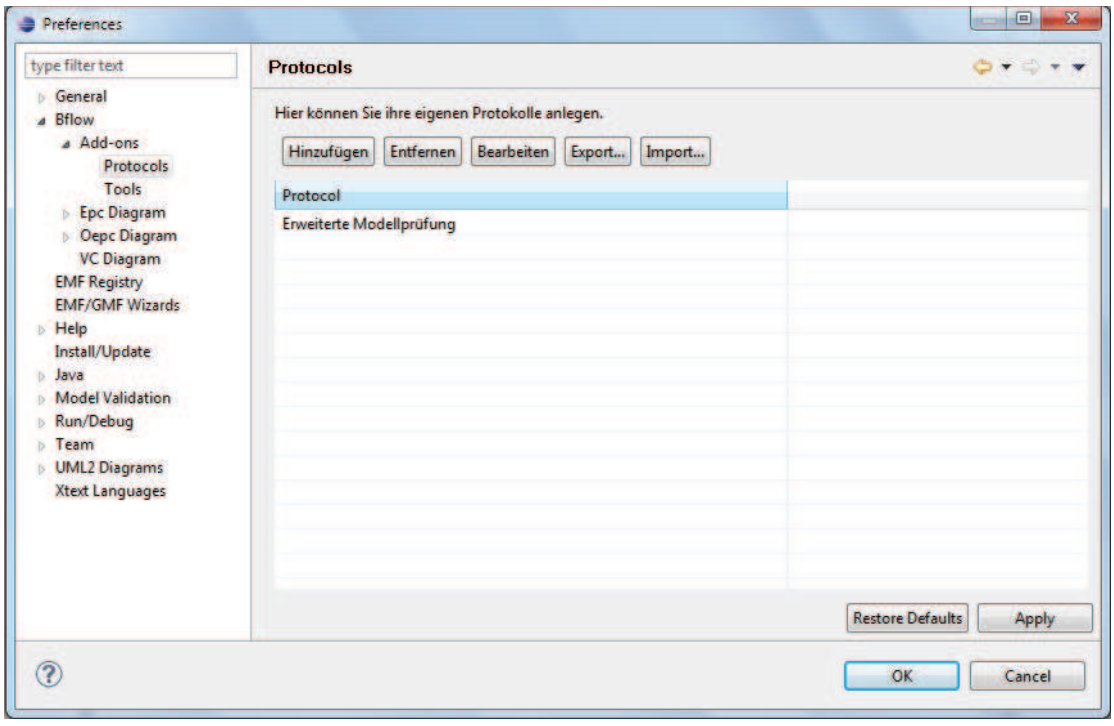
Anmerkung:
Die Aufrufparameter sind Argumente, mit denen das Programm gestartet wird. Hier können Sie prinzipiell alle beliebigen Werte eintragen. Daneben gibt es von bflow* eine Handvoll Schlüsselwerte, die stets mit „$“ beginnen. Im obigen Fall wird bewirkt, dass der Ausdruck „$source“ durch den Pfad der exportierten Modelldatei ersetzt wird.
Wenn bflow* das Programm startet, dann mit folgendem Aufruf:
„java –jar D:\bflow\MTG-Sim.jar –f:D:\bflow\tmp\modelexport.xml“

Eine Liste mit allen reservierten Programmparametern finden Sie am Ende des Kapitels.

Schließen Sie nun den Dialog mit einem Klick auf „OK“. Sie werden dann sehen, dass das Programm in die ursprüngliche Liste mit aufgenommen wurde.

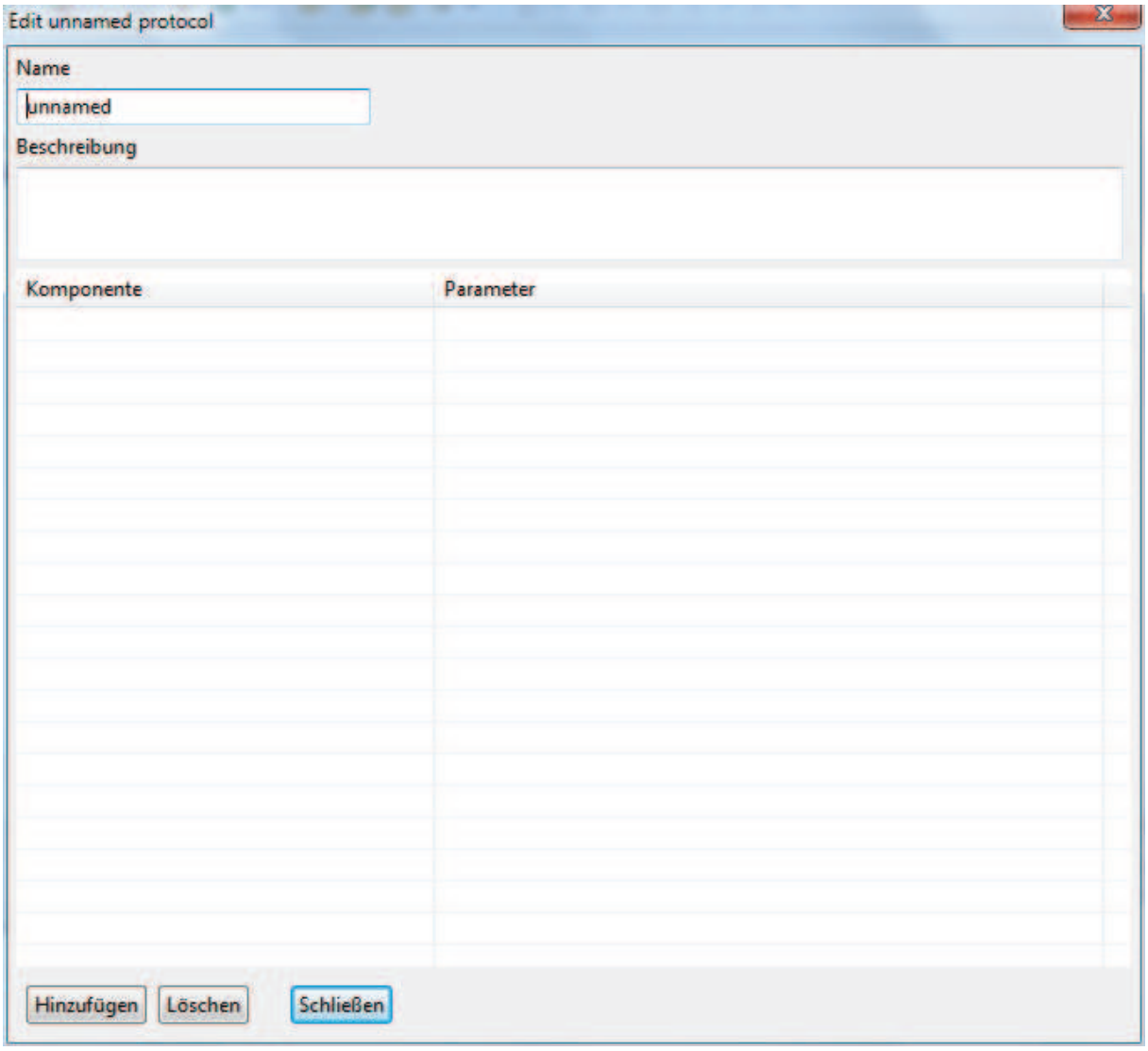


Wählen Sie nun den Unterpunkt „Protocols“ aus. Sie sehen das folgende Fenster:



In diesem Fenster sehen Sie alle bereits installierten Protokolle. Sollten Sie sich nicht sicher sein, was ein installiertes Protokoll macht, so halten Sie den Mauscursor darüber. Nach kurzer Zeit erscheint ein Tooltip, der Ihnen einen kleinen Beschreibungstext anzeigt.

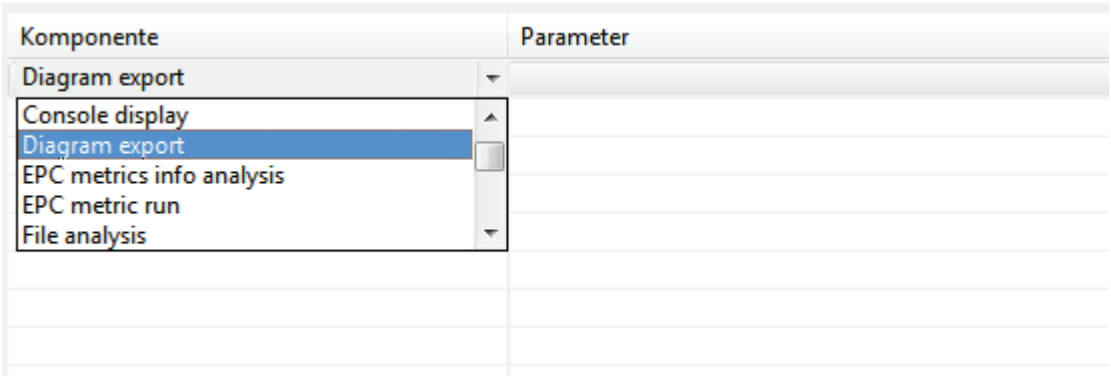
Klicken Sie auf den Button „Hinzufügen“. Es öffnet sich der folgende Dialog:



Geben Sie zuerst den Namen des Protokolls ein. Dieser ist beliebig. Der hier eingegebene Name wird auch zur Darstellung im Bflow*-Add-on-Menü benutzt. Schreiben Sie hier zum Beispiel „MTG-Simulator“.

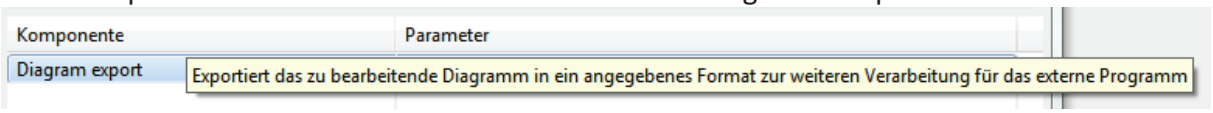
Der Beschreibungstext ist ebenfalls beliebig, sollte aber aussagekräftig sein. Geben Sie zum Beispiel folgenden Text ein:
„Anhand von gegebenen Wahrscheinlichkeiten und Ausführungszeiten simuliert das Programm einen Prozessablauf und misst dabei zum einen, wo die Simulation endete und zum anderen nach welcher Zeit.“

Kommen wir nun zur Einrichtung der Komponenten:
Klicken Sie auf „Hinzufügen“. Sie sehen, dass in der Tabelle nun eine Zeile mit „ – select –“ erscheint. Wenn Sie dieses Feld anklicken, so öffnet sich eine Drop-Down-Liste.



Wählen Sie hier den „Diagram export“. Wenn Sie anschließend in das Parameterfeld klicken, dann öffnet sich ebenfalls eine Drop-Down-Liste. In dieser Liste können Sie auswählen, in welchem Format bflow* das Modell exportieren soll. Hier finden Sie alle Formate, die Sie auch über die Bflow*-Modell-Export-Funktion verwenden können. Wählen Sie hier das „EPCTools EPML-Datei“-Format.

Jede Komponente besitzt eine Beschreibung. Wenn Sie sich nicht sicher sind, was eine Komponente macht, so halten Sie den Mauscursor über die Komponente und warten Sie bis der Tooltip erscheint. In diesem wird Ihnen die Bedeutung der Komponente kurz erläutert.


Komponente
Parameter
Diagram export
Exportiert das zu bearbeitende Diagramm in ein angegebenes Format zur weiteren Verarbeitung für das externe Programm

Klicken Sie nun ein weiteres Mal auf „Hinzufügen“.
Wählen Sie jetzt genauso wie oben die „Tool Run“-Komponente. Öffnen Sie anschließend die Drop-Down-Liste im Parameterfeld und wählen Sie hier den „MTG-Simulator“.

Hinweis:
In der Liste des Parameterfelds erscheinen nur die Programme, die der bflow*-Add-on-Schnittstelle bekannt sind. Sollte ein von Ihnen gewünschtes Programm hier nicht erscheinen, so prüfen Sie, ob es auch korrekt im Menüpunkt unter „Bflow“ → „Add-ons“ → „Tools“ eingetragen ist. Schauen Sie sich dazu nochmal das Beispiel des MTG-Simulators an, das diesen Vorgang beschreibt.

Klicken Sie nun erneut auf „Hinzufügen“ und wählen Sie die Komponente „File analysis“. Diese Komponente analysiert Dateien, die von dem gestarteten Programm erstellt werden. Bei dieser Analyse werden aber nicht alle Informationen verarbeitet, sondern nur die, die ein bestimmtes Präfix und eine festgelegte Struktur besitzen. Weitere Informationen dazu finden Sie in der Komponentenbeschreibung.
Diese Komponente wird somit benötigt, da der MTG-Simulator seine Ergebnisse in eine Datei schreibt. Schreibt ein Programm seine Ergebnisse in die Prozessumgebung, so müsste man die „Shell analysis“-Komponente auswählen. Hier ist das aber nicht der Fall.

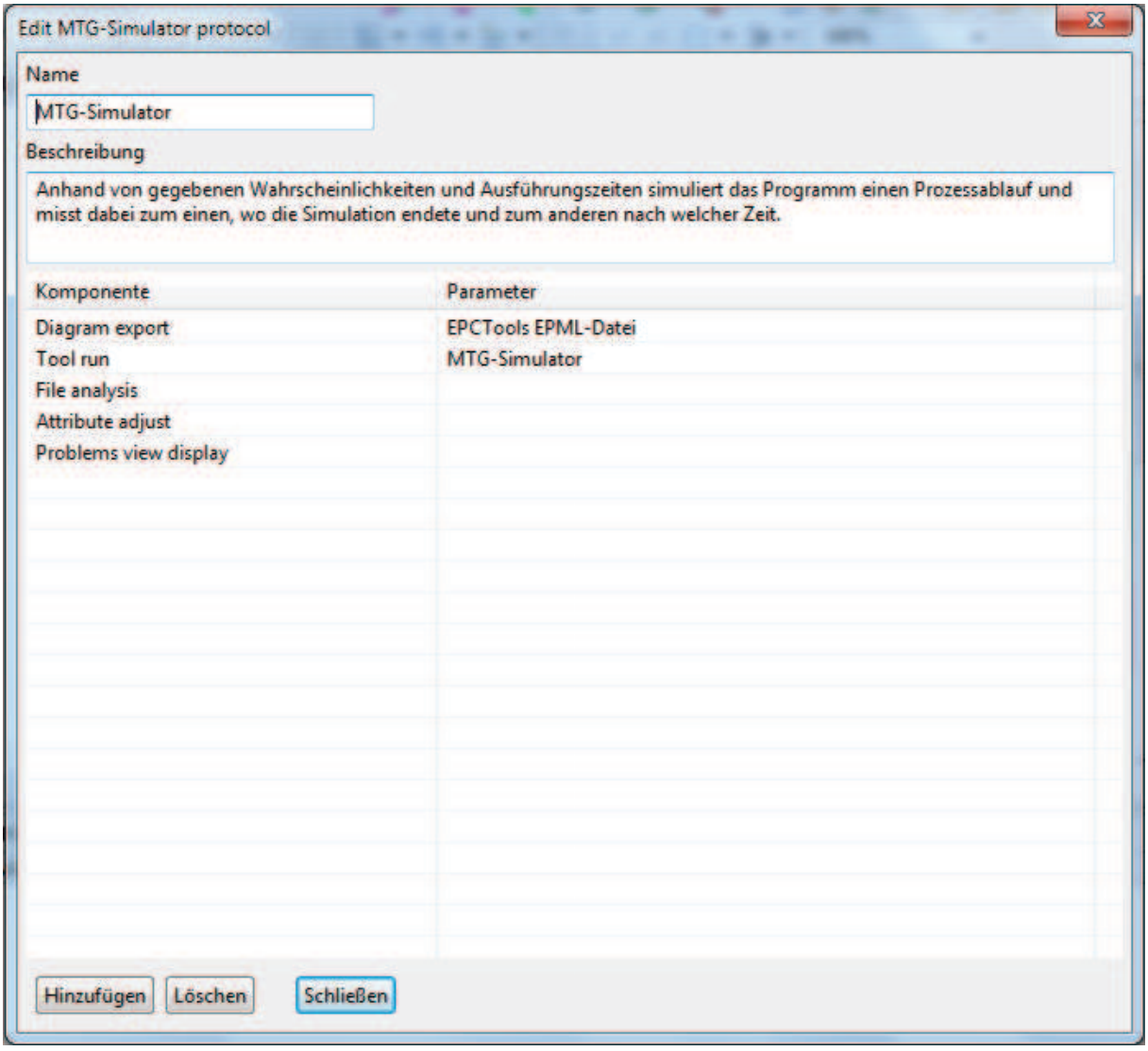
Wenn Sie nun in das Parameterfeld klicken, so merken Sie, dass nichts passiert. Das liegt daran, dass nicht jede Komponente Parameter benötigt. Falls eine Komponente ohne Parameter auskommt, so ist das entsprechende Feld nicht veränderbar.

Als nächstes benötigen wir nun die „Attribute adjust“-Komponente. Fügen Sie eine neue durch einen Klick auf „Hinzufügen“ in das Protokoll ein und wählen als Typ „Attribute

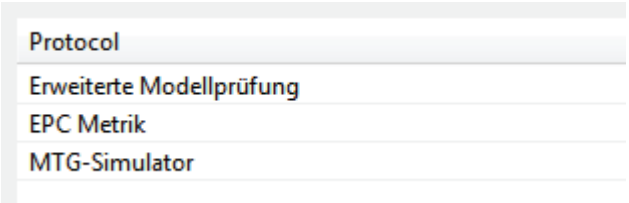
adjust“. Diese Komponente durchsucht die analysierten Meldungen nach Kommandos, um Attribute anzulegen oder anzupassen. Parameter werden hier ebenfalls nicht benötigt.

Kommen wir nun zur letzten Komponente. Fügen Sie eine neue hinzu und wählen als Typ „Problems view display“. Diese Komponente erzeugt Meldungen auf der Bflow*-Problems-View aus den analysierten Programminformationen. Parameter werden nicht benötigt.

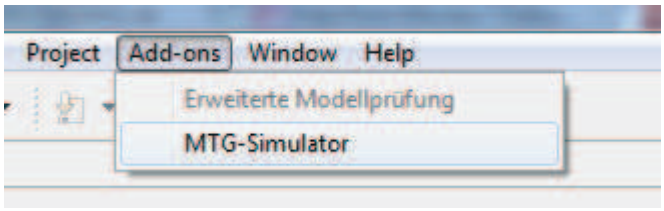
Ihr Protokoll sollte nun wie folgt aussehen:



Mit einem Klick auf „Schließen“ wird das Protokoll gespeichert und erscheint nun in der Protokollliste.



Das Protokoll ist nun sofort (ohne Neustart) einsatzbereit und über das Add-on-Menü ausführbar.



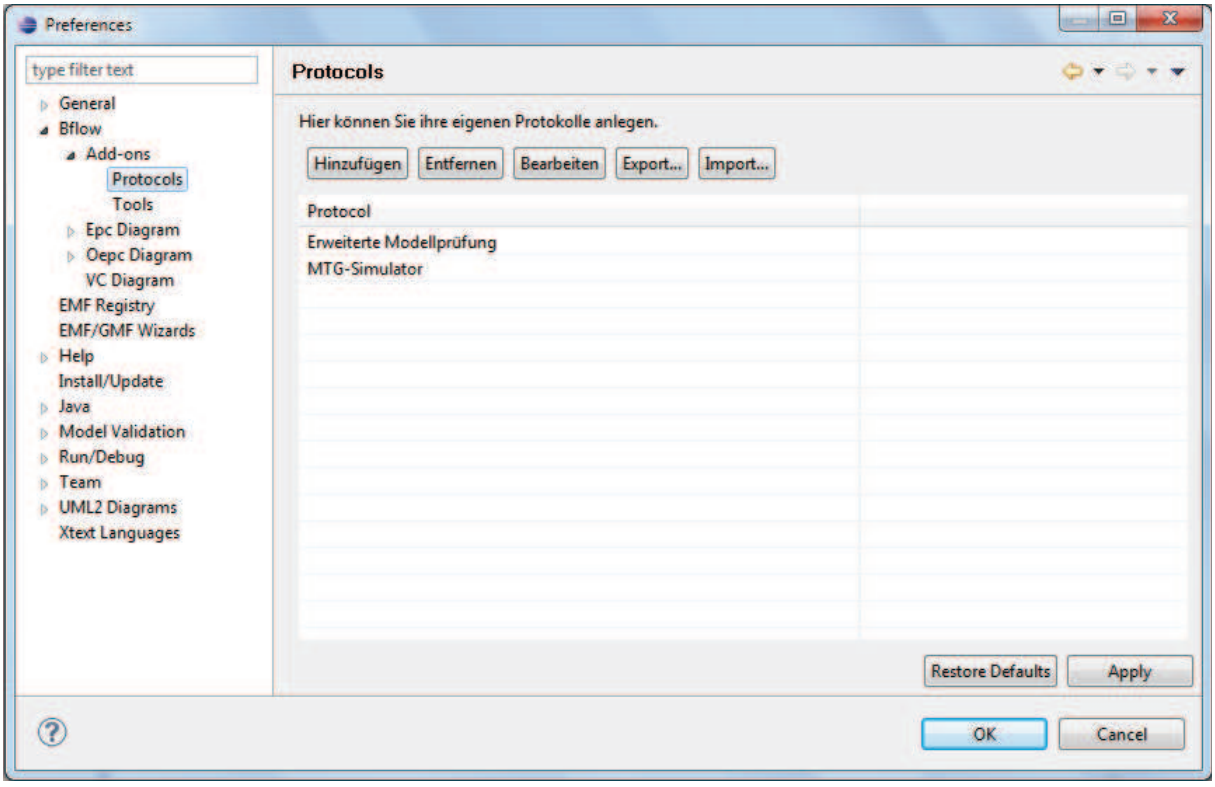
Ein eigenes Prolog-Programm einrichten

Der folgende Abschnitt wird Ihnen erläutern, wie Sie mittels der Add-on-Schnittstelle Ihr eigenes Prolog-Programm einbinden können. Voraussetzung dafür ist, dass Sie den SWI-Prolog-Interpreter installiert haben. Wie das funktioniert, ist ausführlich im Benutzerhandbuch beschrieben.

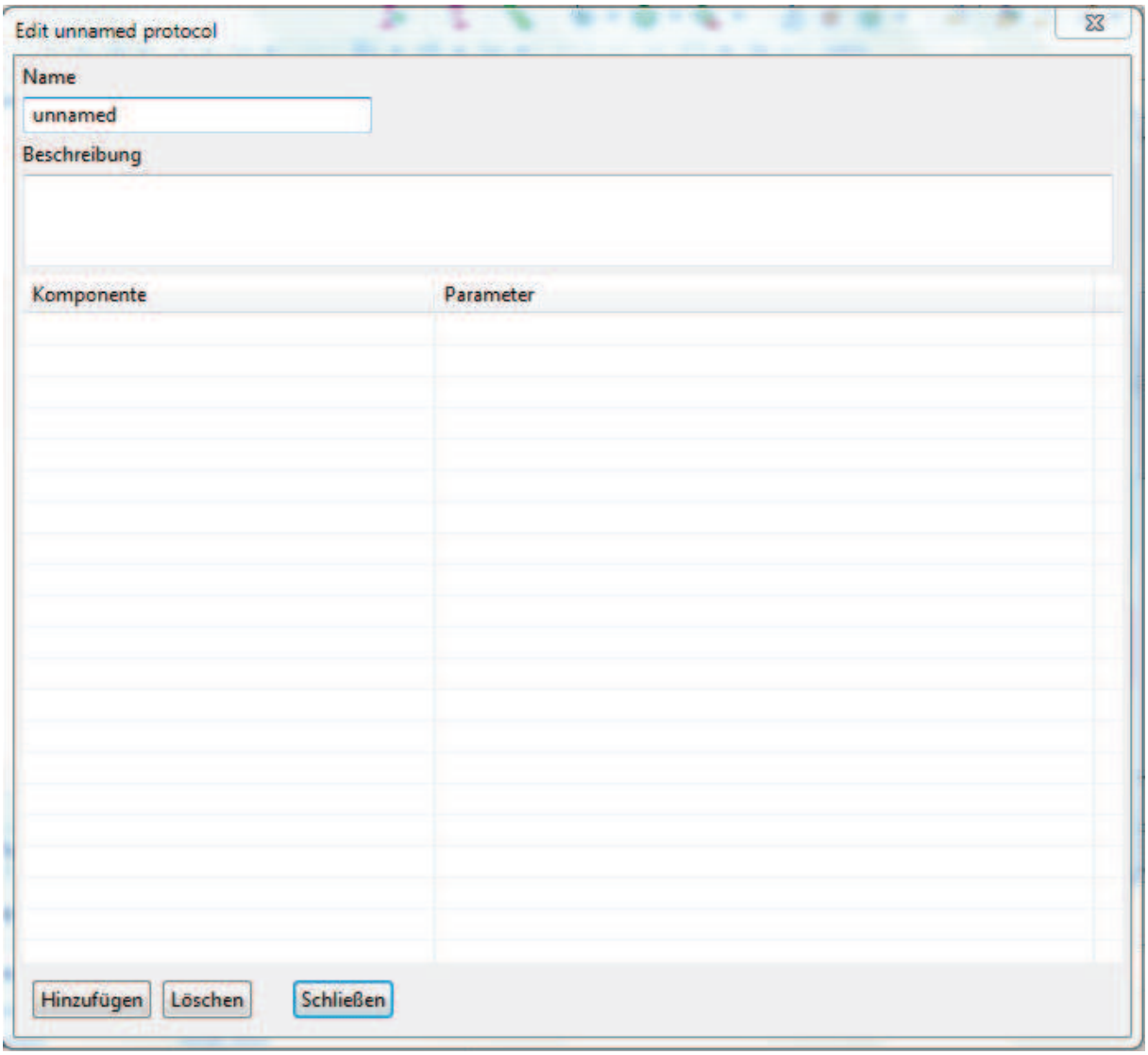
Wir gehen davon aus, dass Sie ein Prolog-Programm irgendwo auf der Festplatte gespeichert haben. Für unser Beispiel heißt das Programm „konvertierungen.pl“. Dieses Programm wandelt das Modell und die eingebetteten Elemente in eine andere Struktur. Diese neue Struktur wird anschließend in die Prozessumgebung geschrieben.

Da Prolog von Haus aus schon für bflow* vorkonfiguriert ist, müssen Sie es nicht als externes Add-on-Programm installieren. Sie müssen lediglich ein geeignetes Protokoll anlegen.

Öffnen Sie dazu das Menü „Window“ → „Preferences“ und anschließend den Unterpunkt „Bflow“ → „Add-ons“ → „Protocols“. Sie sollten das folgende Fenster sehen:



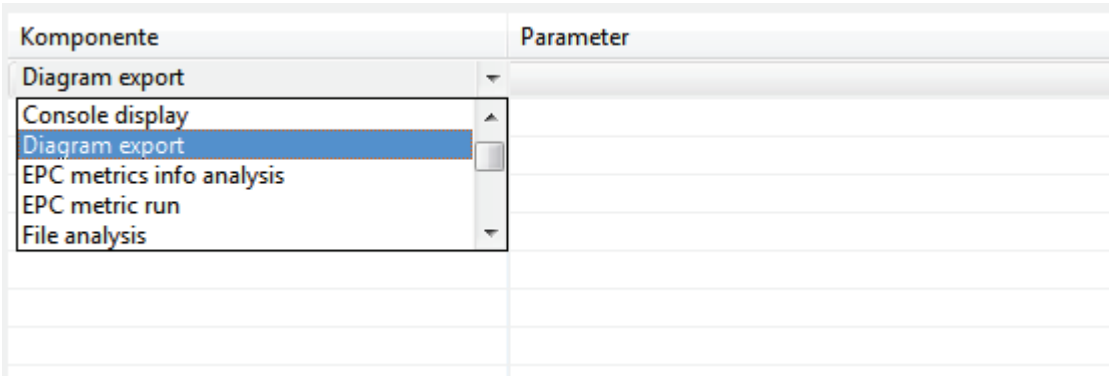
Klicken Sie auf den Button „Hinzufügen“. Es öffnet sich nun der folgende Dialog:



Geben Sie als Namen „Prolog Konvertierung“ ein. Der Name des Protokolls ist prinzipiell frei wählbar. Schreiben Sie anschließend einen kleinen Beschreibungstext. Dieser könnte wie folgt lauten:
„Wandelt die Diagrammstruktur und dessen Elemente in ein anderes Format um.“

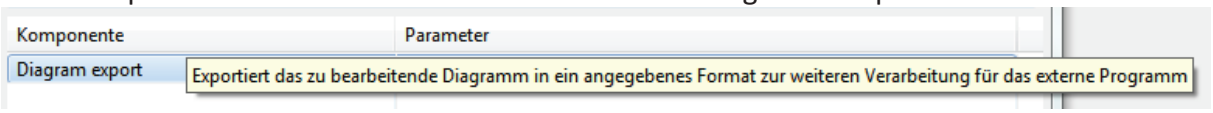
Der Beschreibungstext ist beliebig, sollte aber aussagekräftig sein.

Kommen wir nun zur Einrichtung der Komponenten:
Klicken Sie auf „Hinzufügen“. Sie sehen, dass in der Tabelle nun eine Zeile mit „ – select –“ erscheint. Wenn Sie dieses Feld anklicken, so öffnet sich eine Drop-Down-Liste.



Wählen Sie hier den „Diagram export“. Wenn Sie anschließend in das Parameterfeld klicken, dann öffnet sich ebenfalls eine Drop-Down-Liste. In dieser Liste können Sie auswählen, in welchem Format bflow* das Modell exportieren soll. Hier finden Sie alle Formate, die Sie auch über die Bflow*-Modell-Export-Funktion verwenden können. Wählen Sie hier das „Prolog-Datei“-Format.

Jede Komponente besitzt eine Beschreibung. Wenn Sie sich nicht sicher sind, was eine Komponente macht, so halten Sie den Mauscursor über die Komponente und warten Sie bis der Tooltip erscheint. In diesem wird Ihnen die Bedeutung der Komponente kurz erläutert.


Komponente
Parameter
Diagram export
Exportiert das zu bearbeitende Diagramm in ein angegebenes Format zur weiteren Verarbeitung für das externe Programm

Klicken Sie nun ein weiteres Mal auf „Hinzufügen“.
Wählen Sie jetzt genauso wie oben die „Prolog run“-Komponente. Klicken Sie nun in das Parameterfeld. Diese Komponente benötigt im Wesentlichen zwei Parameter: den Pfad zum Prologprogramm und zusätzliche Angaben zur Steuerung des Prologprogramms. Geben Sie nun den folgenden Text ein: „-pl:D:\bflow\pls\konvertierungen.pl –p:x“

Der Parameter „-pl“ gibt an, wo die Datei liegt. Korrigieren Sie ggf. den Pfad für Ihre Situation. Mit dem Parameter „-p“ lassen sich zusätzliche Programmoptionen übergeben. Da dieses Programm keine weiteren Parameter benötigt, reicht hier einfach ein „x“.

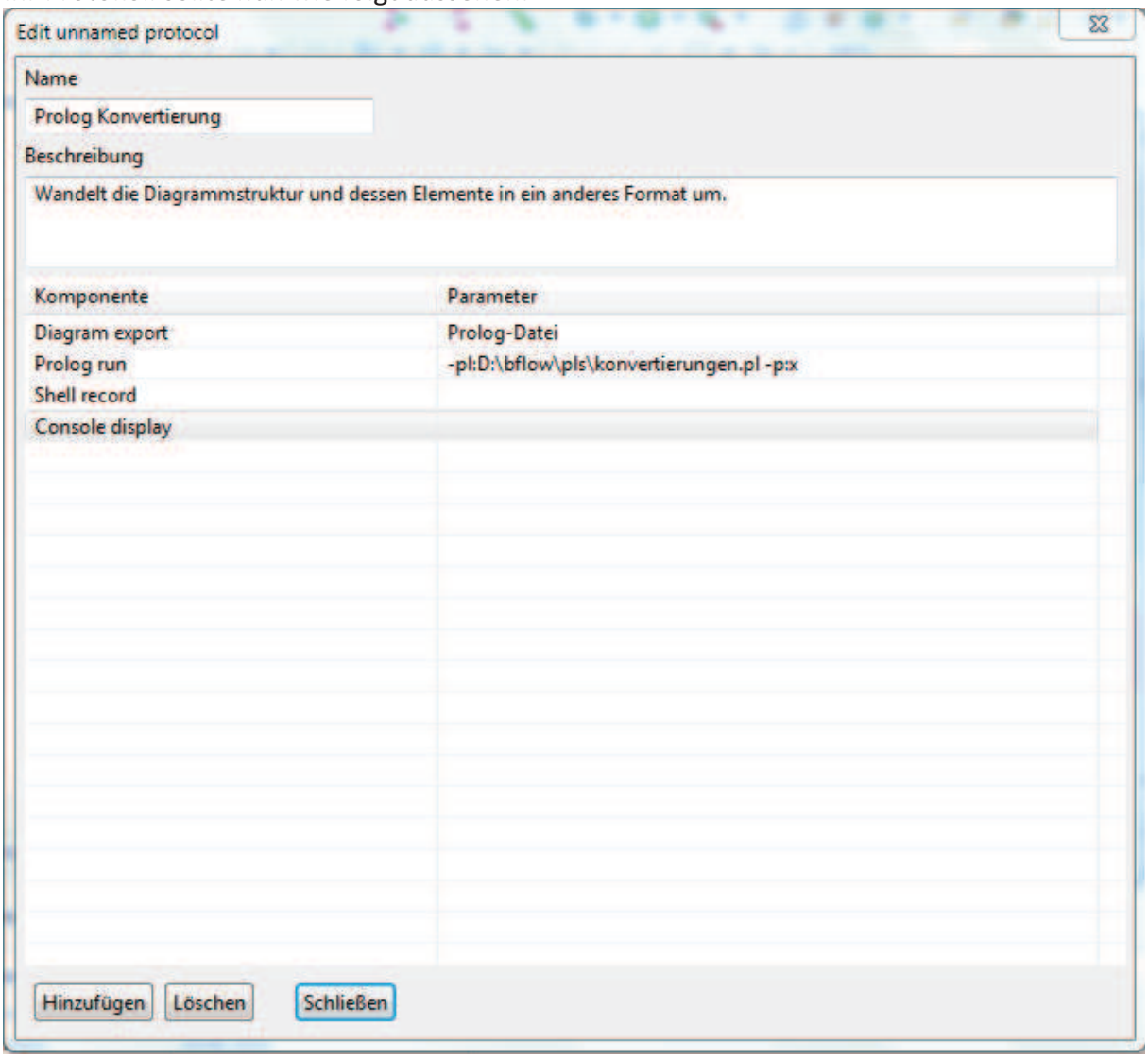
Hinweis:
Wie Sie am Beispiel der erweiterten Modellprüfung sehen können, gibt es für die Parameterliste reservierte Schlüsseldeklarationen, die mit einem „$“ beginnen. Eine vollständige Liste dieser Schlüsselworte und deren Verwendung finden Sie am Ende des Kapitels.

Klicken Sie nun erneut auf „Hinzufügen“ und wählen Sie die Komponente „Shell record“. Diese Komponente zeichnet die Ausgaben des Programms auf der dazugehörigen Prozesslaufzeitumgebung auf. Im Vergleich zur „Shell analysis“-Komponente erfolgt hier keine Analyse der aufgezeichneten Informationen! Weitere Informationen dazu finden Sie in der Komponentenbeschreibung.
Diese Komponente wird somit benötigt, da das Prologprogramm seine Ergebnisse in diese Umgebung (auch Shell genannt) schreibt. Speichert ein Programm seine Ergebnisse in einer Datei, so müsste man die „File record“-Komponente auswählen. Hier ist das aber nicht der Fall.

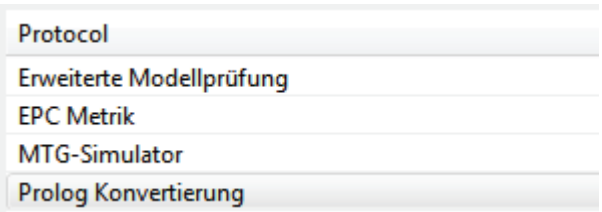
Wenn Sie nun in das Parameterfeld klicken, so merken Sie, dass nichts passiert. Das liegt daran, dass nicht jede Komponente Parameter benötigt. Falls eine Komponente ohne Parameter auskommt, so ist das entsprechende Feld nicht veränderbar.

Kommen wir nun zur letzten Komponente. Fügen Sie eine neue hinzu und wählen als Typ „Console display“. Diese Komponente gibt gesammelte Informationen auf der Add-on-Konsole aus. Parameter werden nicht benötigt.

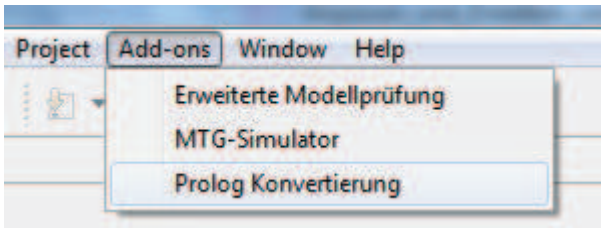
Ihr Protokoll sollte nun wie folgt aussehen:



Mit einem Klick auf „Schließen“ wird das Protokoll gespeichert und erscheint nun in der Protokollliste.

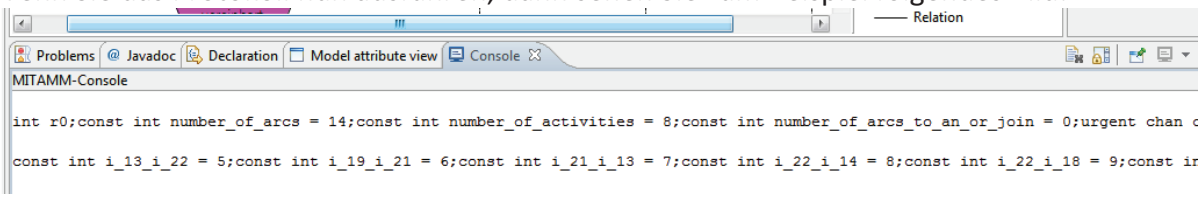


Das Protokoll ist nun sofort (ohne Neustart) einsatzbereit und über das Add-on-Menü ausführbar.



Anmerkung:
Falls Sie noch nicht wissen, wo sich die Add-on-Konsole befindet, so brauchen Sie sich nicht zu sorgen. Die „Console display"-Komponente ist so eingestellt, dass sie die Add-on-Konsole in bflow* automatisch hervorholt, falls diese noch nicht angezeigt wird.

Wenn Sie das Protokoll nun ausführen, dann sehen Sie zum Beispiel folgendes Bild:


Relation
Problems
Javadoc
Declaration
Model attribute view
Console
MITAMM-Console
int r0;const int number_of_arcs = 14;const int number_of_activities = 8;const int number_of_arcs_to_an_or_join = 0;urgent chan c
const int i_13_i_22 = 5;const int i_19_i_21 = 6;const int i_21_i_13 = 7;const int i_22_i_14 = 8;const int i_22_i_18 = 9;const ir

Programmparameter

In dem folgenden Abschnitt werden die reservierten Parameter für Programmaufrufe erläutert. Wenn Sie sich nicht sicher sind, wofür Sie diese Parameter verwenden können, so lesen Sie nochmal auf Seite 16 nach.

Parameter	Bemerkung
$source	Ersetzt beim Aufruf des Programms das Schlüsselwort durch die Datei, die bearbeitet werden bzw. als Input dienen soll. Beispiel: „-f:$source" → „-f:C:\bflow\lagerverwaltung.epml"
$file1, $file2, … $file10, …	Werden im Verlauf eines Protokolls mehrere Dateien erzeugt, die während dessen Ausführung benötigt werden, so können sie wie links dargestellt, referenziert werden. Angenommen Ihr Protokoll beinhaltet drei externe Programmaufrufe. Programm 1 und Programm 2 erzeugen jeweils eine Datei. Programm 3 muss mit diesen Dateien und der Quelldatei arbeiten. Dann sollte Programm 3 folgende Parameter besitzen: „$source $file1 $file2"
$bflow_temp	Dieser Parameter liefert einen Zeiger auf ein temporäres Verzeichnis. Beispiel: „$bflow_temp" → „C:\Windows\temp\"
$install_dir	Dieser Parameter liefert einen Zeiger auf das Verzeichnis, in dem das Programm installiert ist. Beispiel:

„$install_dir“ → „D:\bflow\addons\epcmetrik\“

Prologparameter

Für die Ausführung der „Prolog run“-Komponente existieren reservierte Schlüsselwörter, die das Verhalten des Prolog-Interpreters steuern. Wenn Sie sich nicht sicher sind, wofür Sie diese Parameter verwenden können, so lesen Sie erneut auf Seite 23 nach.

Parameter	Bemerkung
-pl:	Gibt an, welches Prolog-Programm verwendet werden soll. Die Angabe muss entweder ein absoluter Pfad oder eine Programmkonstante sein. Beispiel: (1) -pl:C:\bflow\prolog\myEval.pl (2) -pl:$ExtendedModelValidation
-p:	Gibt eine Liste von Parametern an, die zum Start des Prolog-Programms übergeben werden soll. Die Angabe ist optional und kann Programmkonstanten beinhalten. Beispiel: (1) -p:simple,nonX (2) -p:$bflow_rules
-s:	Gibt den Startbefehl für das Prologprogramm an. Fehlt dieser Parameter, so wird das Prologprogramm mit dem Aufruf „bflow_query(…)“ gestartet. Soll Ihr Programm anders gestartet werden, so geben Sie den entsprechenden Aufruf ohne Klammern hier an. Beispiel: (1) –s:a (2) –s:do_request
-e:	Gibt den Terminierungsbefehl für das Prologprogramm an. Fehlt dieser Parameter, so wird erwartet, dass das Prologprogramm mit der Shell-Ausgabe „#query_end#“ terminiert wird. Wird Ihr Programmende durch eine andere Ausgabe angezeigt, so geben Sie den Befehl hier ohne Klammern an. Beispiel: (1) –e:STOP (2) –e:#request_finished#

Es folgt nun eine Erläuterung der Konstanten:

Konstante	Bemerkung	Programm	Parameter
$ExtendedModelValidation	Nur für SWI-Prolog! Benutzt das von bflow*ausgelieferte Programm zur erweiterten Modellprüfung.	X	
$bflow_rules	Nur für SWI-Prolog! Kennzeichnet eine Parameterliste, die die zur Prüfung eingestellten Regeln übergibt. Diese Regeln können in den Programmeinstellungen eingestellt werden.		X

Startbefehlskonvention

Damit ein Prologprogramm von bflow* korrekt gestartet werden kann, gibt es eine Festlegung für die Syntax des Startbefehls:

STARTBEFEHL (SPRACHE, [PARAMETERLISTE]).

Als Startbefehl gilt, sofern Sie keinen eigenen festgelegt haben, die Zeichenkette „bflow_query“. Danach folgt ein Sprachkürzel. Dies könnte „de“ oder „en“ usw. sein. Dieses wird automatisch festgelegt und entspricht der eingestellten bflow*-Programmsprache. Abschließend wird eine Liste von Parametern angehängt. Diese Parameter entsprechen den Programmparametern, die Sie mittels „-p:“ bei der Einrichtung des Prologprogramms festlegen.

Angenommen, Sie haben ein Protokoll mit folgendem Prologaufruf festgelegt:
„-pl:D:\bflow\pls\myEval.pl –p:short,lang,xtract –s:query –e:query_end”.

Mittels dieser Parameter wird der Prologaufruf wie folgt gestaltet:
„query(de,[short,lang,xtract]).“

Lautet Ihr Prologaufruf wie folgt: “-pl:D:\bflow\pls\myEval.pl –p:x“. Dann wird Prolog derart gestartet: „bflow_query(de,[x]).“

Add-on-Komponenten

In dem folgenden Abschnitt werden die verfügbaren Add-on-Komponenten erläutert.

Komponente	Beschreibung
Attribute adjust	Verwenden Sie ein Protokoll, das u.a. Modellattribute verändert, so müssen Sie auch diese Komponente in Ihrem Protokoll verwenden. Es sorgt dafür, dass die neuen und geänderten Attribute von bflow* erkannt und das Modell entsprechend angepasst wird. Damit diese Erkennung korrekt funktioniert, muss die Information aus dem externen Programm einer Formkonvention entsprechen. Diese Formkonvention ist im nächsten Abschnitt erläutert.

	Diese Komponente können Sie nur nach der „File analysis“ oder „Shell analysis“-Komponente verwenden. Sie benötigt keine Parameter.
Console display	Möchten Sie, dass die Informationen aus dem externen Programm lediglich auf der Add-on-Konsole angezeigt werden, so können Sie dies mit dieser Komponente einstellen. Sie nimmt dabei keine Veränderungen an diesen Informationen vor und gibt Sie anschließend an die nächste Komponente weiter. Diese Komponente können Sie nur nach der „File analysis“, „Shell analysis“ oder „Shell record“-Komponente verwenden. Sie benötigt keine Parameter.
Diagram export	Diese Komponente exportiert das zu bearbeitende Diagramm in ein gewünschtes Format. Als Format stehen alle installierten Exportskripte zur Verfügung (Siehe Seite 9). Im Normalfall ist dies stets die erste Komponente eines Protokolls sein! Daher kann diese Komponente keiner anderen Komponente folgen. Als Parameter stehen in einer Drop-Down-Liste alle installierten Exportskripte zur Verfügung.
File analysis	Nutzen Sie diese Komponente, wenn ihr externes Programm seine Ergebnisse in eine Datei schreibt. Die Komponente analysiert diese Datei nach Informationen, die für bflow* verwendbar sind. Dies können Meldungen für die Problems-View oder Attributänderungen sein. Damit diese Erkennung korrekt funktioniert, muss die Information aus dem externen Programm einer Formkonvention entsprechen. Diese Formkonvention ist im nächsten Abschnitt erläutert. Diese Komponente können Sie nur nach der „Tool run“ oder „Prolog run“-Komponente verwenden. Sie benötigt keine Parameter. *Beachten Sie unbedingt den Abschnitt zur **Arbeit mit externen Dateien**!*
File record	Wenn Sie lediglich die Ergebnisse, die das externe Programm in eine Datei schreibt, ohne Analyse aufzeichnen wollen, so verwenden Sie diese Komponente in Ihrem Protokoll. Da keine Analyse stattfindet, müssen auch keine Konventionen eingehalten werden. Dies kann aber je nach Ausgabe des externen Programms zur Darstellung von kryptischen Zeichen führen. Wollen Sie, dass die Informationen auf der Add-on-Konsole angezeigt werden, so fügen Sie danach die „Console display“-Komponente ein.

	Sie können diese Komponente nach der „Tool run“ oder „Prolog run“-Komponente in Ihrem Protokoll verwenden. Sie benötigt keine Parameter. *Beachten Sie unbedingt den Abschnitt zur **Arbeit mit externen Dateien**!*
Problems view display	Wenn Sie möchten, dass das Ergebnis der Analyse der Daten des externen Programms in der Problems-View angezeigt wird, so verwenden Sie diese Komponente in Ihrem Protokoll. Dabei werden nur die Meldungen angezeigt, die analysiert werden konnten. Entsprechen die Meldungen aus dem externen Programm nicht der Formkonvention, so können auch keine Meldungen angezeigt werden. Diese Formkonvention ist im nächsten Abschnitt erläutert. Diese Komponente kann nach der „Shell analysis“-, „File analysis“ oder „Attribute adjust“-Komponenten eingesetzt werden. Wenn Sie diese Komponente verwenden wollen, so sollte sie im Normalfall die Letzte Ihres Protokolls sein. Sie benötigt keine Parameter.
Prolog run	Diese Komponente startet - sofern installiert - den Prolog-Interpreter. Sie kann nach der „Diagram export“- oder „Tool adapter“-Komponente in einem Protokoll platziert werden. Die Komponente benötigt zwingend zwei Parameter: „-pl:“ und „-p“. Wofür diese Parameter sind und welche weiteren es gibt, können Sie auf Seite 25 nachlesen.
Shell analysis	Nutzen Sie diese Komponente, wenn ihr externes Programm seine Ergebnisse in die Prozesslaufzeitumgebung schreibt. Die Komponente analysiert diese Umgebung nach Informationen, die für bflow* verwendbar sind. Dies können Meldungen für die Problems-View oder Attributänderungen sein. Damit diese Erkennung korrekt funktioniert, muss die Information aus dem externen Programm einer Formkonvention entsprechen. Diese Formkonvention ist im nächsten Abschnitt erläutert. Diese Komponente können Sie nur nach der „Tool run“ oder „Prolog run“-Komponente verwenden. Sie benötigt keine Parameter.
Shell record	Wenn Sie lediglich die Ergebnisse, die das externe Programm auf die Prozesslaufzeitumgebung schreibt, ohne Analyse aufzeichnen wollen, so verwenden Sie diese Komponente in Ihrem Protokoll. Da keine Analyse stattfindet, müssen auch keine Konventionen eingehalten werden. Dies kann aber je nach Ausgabe des externen

	Programms zur Darstellung von kryptischen Zeichen führen. Wollen Sie, dass die Informationen auf der Add-on-Konsole angezeigt werden, so fügen Sie danach die „Console display“-Komponente ein. Sie können diese Komponente nach der „Tool run“ oder „Prolog run“-Komponente in Ihrem Protokoll verwenden. Sie benötigt keine Parameter.
Tool adapter	Möchten Sie mehrere externe Programme hintereinander ausführen, so verwenden Sie diese Komponente als Brücke zwischen beiden. Sie prüft, welchen Input das vorhergehende Programm für das nächstfolgende bereitstellt. Es prüft außerdem, ob eine Adaption überhaupt möglich ist und nimmt ggf. Anpassungen vor. Diese Komponente kann nur nach der „Tool run“- oder „Prolog run“-Komponente eingesetzt werden. **Achtung:** Wenn Programm A das richtige Format für Programm B bereits bereitstellt und somit keine Anpassung nötig ist, so wählen Sie als Parameter aus der Drop-Down-Liste „equal“. Der Adapter leitet in diesem Fall nur die Pfadangabe bzw. Prozesslaufzeitumgebungsausgabe weiter. Wenn Sie eine Konvertierung von der Shell-Ausgabe in eine Datei benötigen, so verwenden Sie „shell to file“ als Parameter. Für den umgekehrten Fall „file to shell“.
Tool run	Mit dieser Komponente können Sie beliebige externe Programme innerhalb Ihres Protokolls starten. Bedingung dafür ist, dass sich das Programm innerhalb Ihres Betriebssystems selbstständig starten lässt. Zur Auswahl stehen nur Programme, die für Add-ons registriert sind. Wenn Sie sich nicht sicher sind, ob das für Ihr Programm zutrifft, so lesen Sie bitte auf Seite 15 nach. Der Parameter entspricht dem zu startenden Programm. Diese Komponente kann nur nach der „Diagram export“- oder „Tool adapter“-Komponente ausgeführt werden.

Formkonvention für Analysen und Modifikationen des Modells

In dem folgenden Abschnitt soll erklärt werden, welche Form die Meldungen der externen Programme haben muss, damit bflow* daraus Informationen für die Problems-View oder Attributänderungen gewinnen kann. Diese Meldungen können entweder in einer Datei stehen („File analysis“-Komponente) oder in die Prozesslaufzeitumgebung („Shell analysis“-Komponente) geschrieben werden.

Für Informationen, die ausgewertete werden sollen, gilt **generell**:

a. Informationen werden zeilenweise erfasst. Gibt es innerhalb der Ausgabe einen Zeilenumbruch, so wird der nachfolgende Inhalt in einer neuen Informationseinheit aufgezeichnet.
b. Jede Informationseinheit, die von bflow* verarbeitet werden soll, muss mit dem Präfix „bflow*:“ beginnen. Alle anderen Einheiten werden ignoriert.

Soll aus einer Informationseinheit eine Meldung für die **Problems-View** erzeugt werden, so muss die folgende Struktur erfüllt sein:

[MESSAGE][TYP][ELEMENT ID][MELDUNG][ZUSATZ]#FS#

Message	Gibt den Typ der Meldung an. Ist Programmkonstante und muss erscheinen!
Typ	Muss „INFO“, „WARNING“ oder „ERROR“ sein
Element id	Optional. Wenn vorhanden, dann wird die Meldung mit dem entsprechenden Modellelement verknüpft.
Meldung	Nachricht, die angezeigt werden soll
Zusatz	Optional. Wird nicht ausgewertet

Beispiele:

bflow*:[MESSAGE][INFO][_eah882k094][Beschriftung fehlt!][]#FS#
bflow*:[MESSAGE][ERROR][_fgk19o78h][Element unnötig][]#FS#

Für eine Informationseinheit, die **Attributänderungen** nach sich ziehen soll, gilt:

[ATTRIBUTE][TYP][ELEMENT ID][NAME][WERT]#FS#

Attribute	Gibt den Typ der Meldung an. Ist Programmkonstante und muss erscheinen!
Typ	Muss „ADD“, „SET“ oder „REMOVE“ sein
Element id	ID des dazugehörigen Elements
Name	Name des Attributs

Wert Wert des Attributs

bflow*:[ATTRIBUTE][SET][_eah882k094][Planungskosten][1200]#FS#
bflow*:[ATTRIBUTE][ADD][_fgk19o78h][Gezählte Anfragen][21]#FS#
bflow*:[ATTRIBUTE][REMOVE][_uzkkla182f][Erwartete Durchläufe][]#FS#

Es gibt in bflow* zwei Arten von Attributen: Modell- bzw. Elementattribute und Grafische Attribute. Modell- und Elementattribute sind solche, die Sie dem Modell bzw. jedem Element mittels „Attribut-View“ zuordnen können.
Grafische Attribute sind Schriftart, Schriftgröße, Schriftstil (fett, kursiv, unterstrichen) sowie die Position eines Elements. Wenn Sie diese Eigenschaften verändern möchten, so benötigen Sie als Name des Attributs die folgenden Schlüsselwörter:

$name	Legt den Namen des Modellelements fest
$fontname	Legt die Schriftart fest, die das Modellelement verwendet
$fontsize	Legt die Größe der Schrift fest, die das Modellelement verwendet
$fontstyle	Legt den Schriftstil fest, den das Modellelement verwendet
$location	Bestimmt die Position eines Modellelements im Diagramm. Dabei kann die Position entweder absolut oder relativ vorgegeben werden. Für eine relative Positionierung müssen Vorzeichen verwendet werden.

Eine Ausgabe konnte dann wie folgt aussehen:
bflow*:[ATTRIBUTE][SET][_eah882k094][$name][checked]#FS#
bflow*:[ATTRIBUTE][SET][_eah882k094][$fontname][Tahoma]#FS#
bflow*:[ATTRIBUTE][SET][_eah882k094][$fontsize][16]#FS#
bflow*:[ATTRIBUTE][SET][_eah882k094][$fontstyle][BOLD]#FS#
bflow*:[ATTRIBUTE][SET][_eah882k094][$location][540,120]#FS#
bflow*:[ATTRIBUTE][SET][_eah882k094][$location][+20,-60]#FS#

Arbeit mit externen Dateien

Wenn Sie mit der „File analysis“- bzw. der „File record“-Komponente arbeiten, so erzeugt ihr externes Programm eine Datei, deren Standort bflow* zunächst nicht kennt. Um bflow* den Pfad zu dieser Datei mitzuteilen, muss ihr externes Programm folgende Konvention einhalten:

a. Das Programm schreibt eine Zeile in die Prozesslaufzeitumgebung. Diese Zeile hat den Präfix „bflow*:“.
b. Nach dem Präfix folgt eine relative oder absolute Pfadangabe zu der Datei, deren Inhalt aufgezeichnet bzw. analysiert werden soll.

Sie können einen **relativen Pfad** nutzen, wenn die Datei im gleichen Verzeichnis liegt wie das ausgeführte Programm.

Beispiel:
Das Programm liegt in „D:\bflow\MTG-Sim.jar“. Es erzeugt eine Datei namens „result.txt“ zur Analyse in dem gleichen Verzeichnis. Somit muss dieses Programm folgende Zeile in die Prozesslaufzeitumgebung schreiben:
„bflow*:./results.txt“

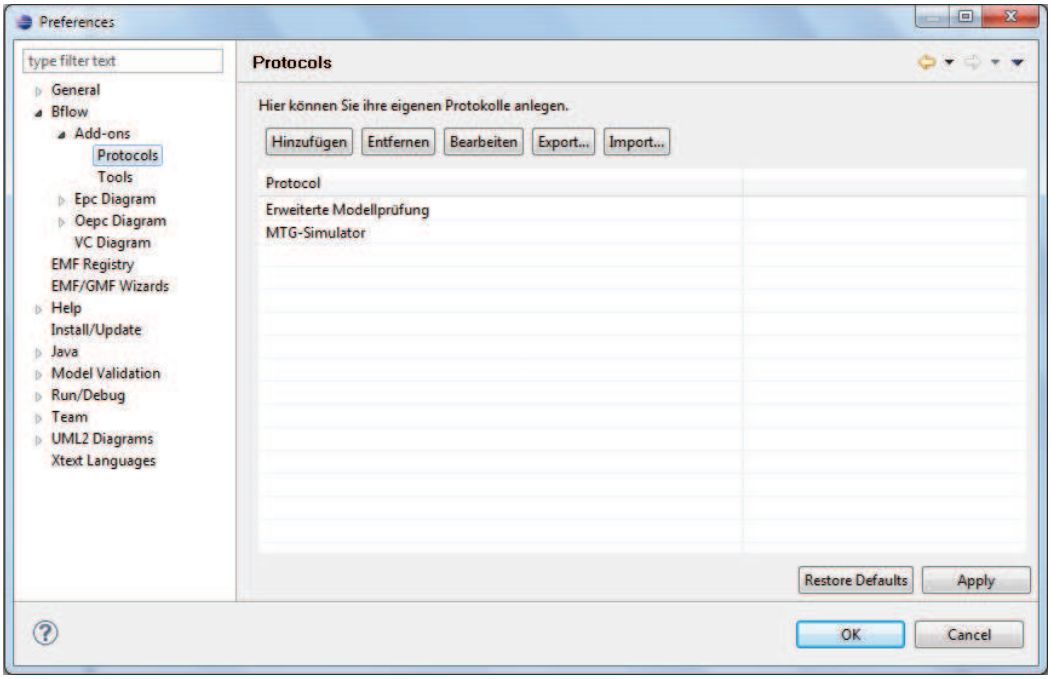
Alternativ können Sie auch hier den **absoluten Pfad** nutzen. In diesem Fall muss das Programm folgende Zeile ausgeben:
„bflow*:D:\bflow\results.txt“

*Liegt die erzeugte Datei nicht in dem gleichen Ordner wie das ausgeführte Programm, so muss es einen **absoluten Pfad** ausgeben!*

Im- und Export von Protokollen

Für den leichteren Austausch von Protokollen zwischen bflow*-Anwendern besitzt die Add-on-Schnittstelle eine Im- und Exportfunktion. Damit können Sie einfach die von Ihnen erstellten Protokolle an andere bflow*-Anwender weitergeben oder deren Protokolle in Ihrer bflow*-Umgebung verwenden.

Öffnen Sie zunächst das Menü „Window“ → „Preferences“ und anschließend den Unterpunkt „Bflow“ → „Add-ons“ → „Protocols“. Sie sollten das folgende Fenster sehen:



Für den **Export** eines Protokolls führen Sie folgende Schritte durch:

Wählen Sie in diesem Fenster das Protokoll aus, das Sie exportieren möchten. Klicken Sie danach auf den Button „Export…“.

Es öffnet sich nun ein Dateidialog. Wählen Sie abschließend den Ort und den Namen des exportierten Protokolls und beenden den Vorgang mit einem Klick auf „OK“.

Die erzeugte Datei besitzt das XML-Format. Aus lizenzrechtlichen Gründen können die Programme, die zur Ausführung des Protokolls nötig sind, nicht integriert werden. Eine separate Installation bzw. Bereitstellung ist daher erforderlich. Dennoch werden die Programmparameter, die das Protokoll verwendet, in das Exportformat integriert. Lesen Sie dazu mehr beim Import eines Protokolls.

Für den **Import** eines Protokolls führen Sie folgende Schritte durch:

Klicken Sie in diesem Fenster auf den Button „Import…“. Es öffnet sich nun ein Dateidialog. Wählen Sie anschließend die XML-Datei aus, die vorher aus einer anderen bflow*-Anwendung heraus exportiert wurde. Schließen Sie den Dialog mit einem Klick auf „OK“.

Wenn es sich um eine korrekt exportierte Datei handelt, dann sehen Sie das Protokoll sofort in Ihrer Protokollliste.

Benötigt ein importiertes Protokoll ein Programm, welches noch nicht für die Add-on-Schnittstelle verfügbar ist, so legt es die Rahmenbedingungen (Name und Parameter) automatisch an. Ein Programm gilt als nicht verfügbar, wenn keines der angemeldeten Programme den gleichen Namen besitzt, wie das vom Protokoll benötigte. In diesem Fall können Sie entweder das Protokoll bearbeiten oder den Pfad des neuen Programms anpassen. Insgesamt ist es jedoch entscheidend, dass die für den Programmaufruf notwendigen Parameter automatisch erzeugt werden. Diese sollten Sie nicht ändern, wenn Sie nicht wissen, was sie bedeuten.